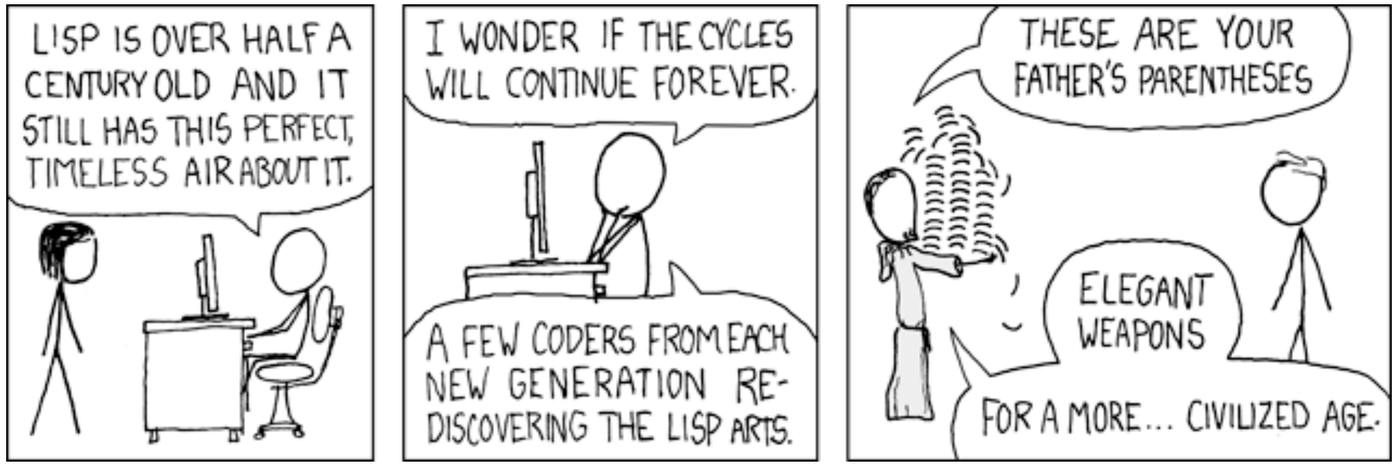


Llama.Lisp: AI First Compiler Framework

Rationale and Design

Sasank Chilamkurthy

Outline

- Why my own compiler framework
- Insides of a compiler
- Principles of my design

This work wouldn’t have been possible without

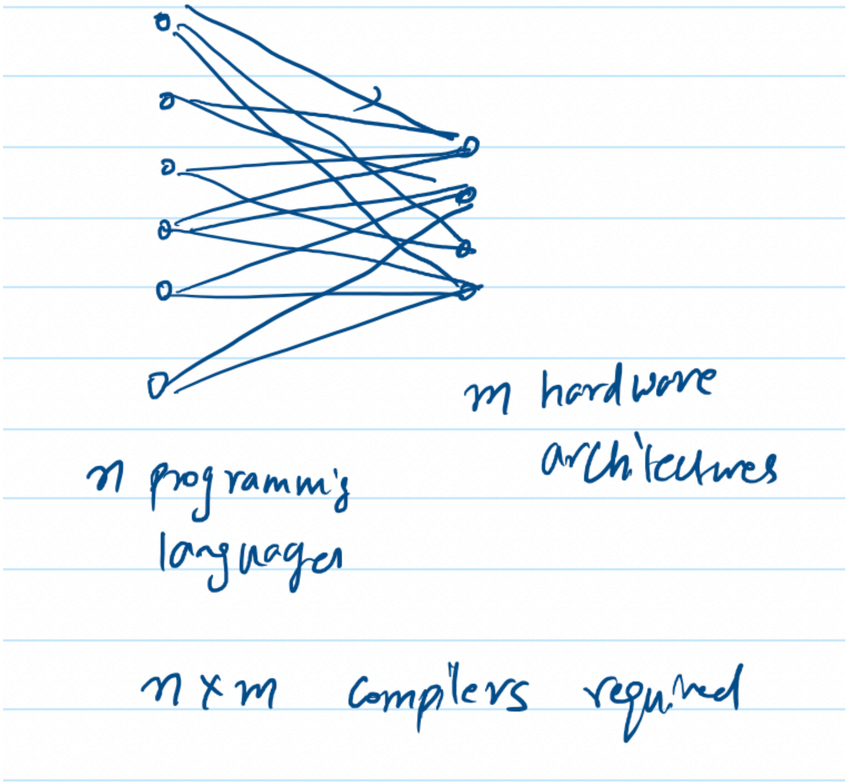
- **Vedanth Padmaraman**
- **Adithya L Bhat**

Why My Own Compiler Framework?

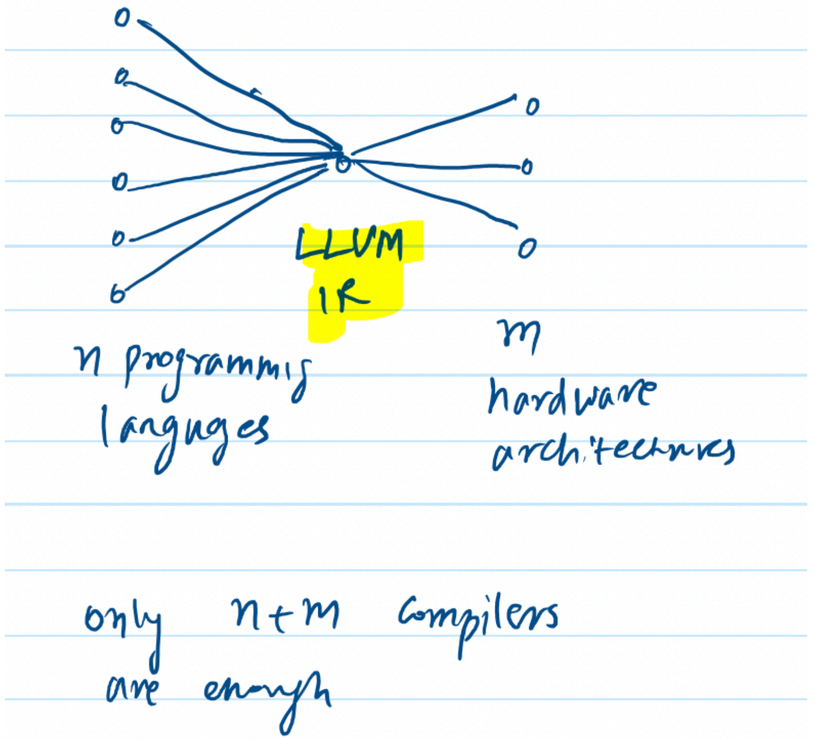
Compilers are Translators

Just like English to Kannada but Rust to x86

Without IR

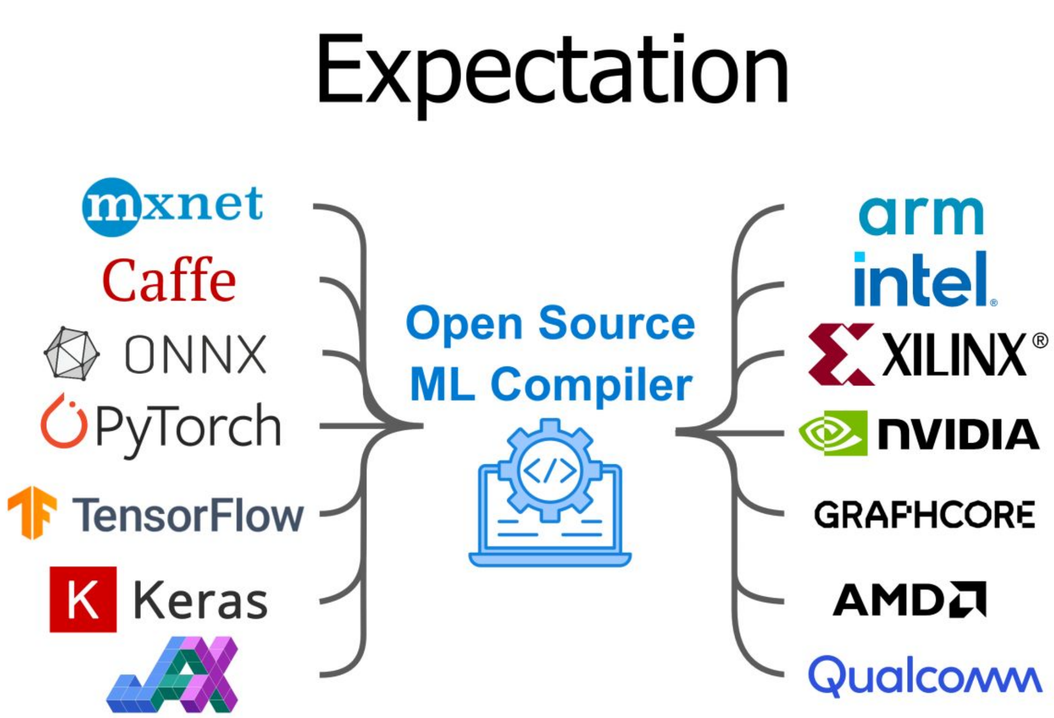


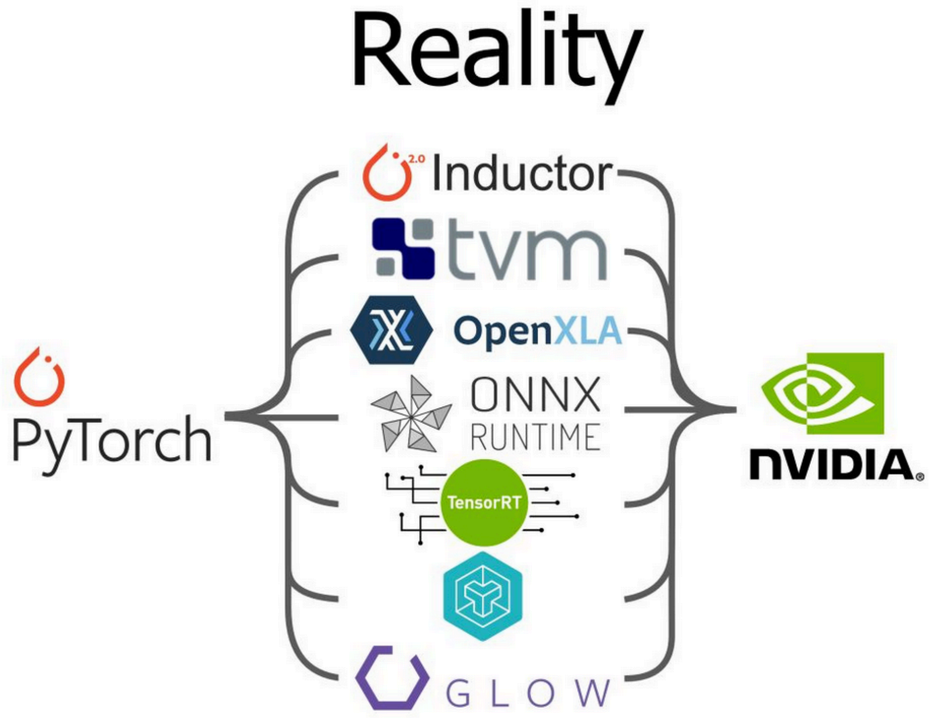
With IR



https://aosabook.org/en/v1/llvm.html

AI Compilers: Expectation vs Reality

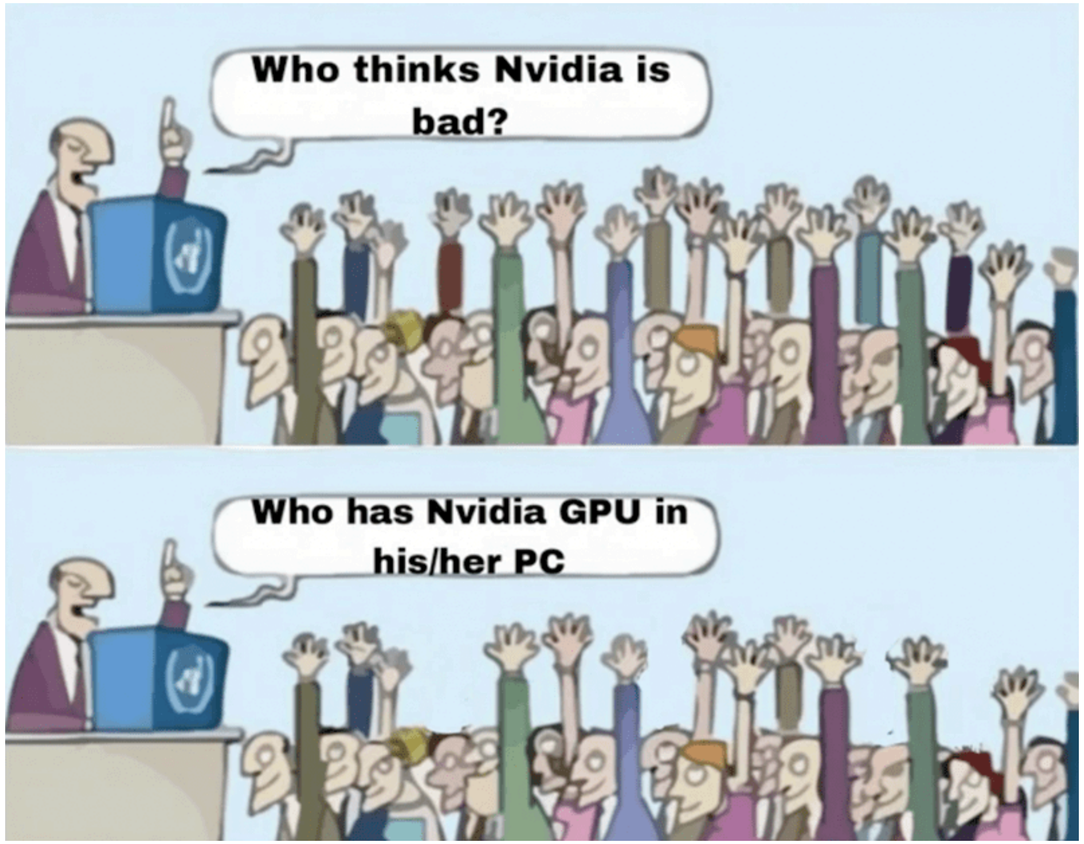




https://x.com/AartBik/status/1782216468006138312

The Consequence

We’re Stuck with Single Vendor



https://www.reddit.com/r/pcmasterrace/comments/14w6ar4/sad_but_true/

GPU Poor

GPUs unaffordable by most people



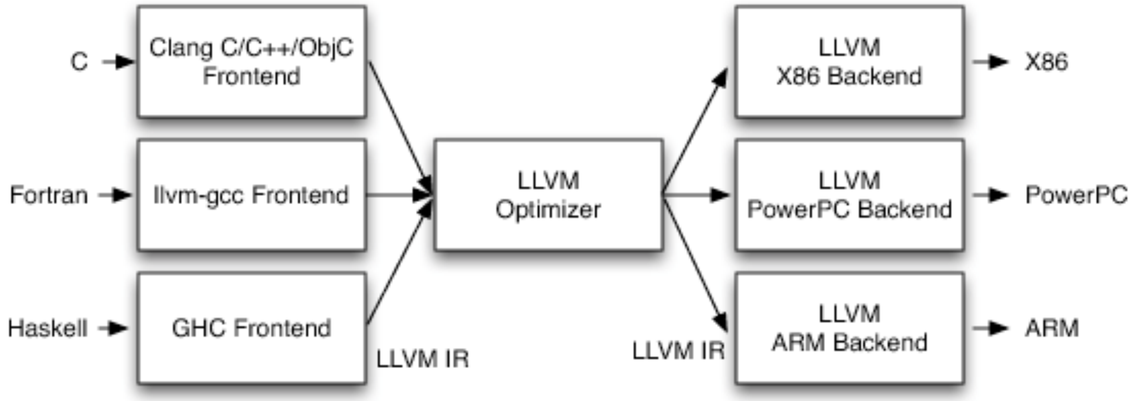
https://x.com/var_epsilon/status/1712301543980925331/photo/1

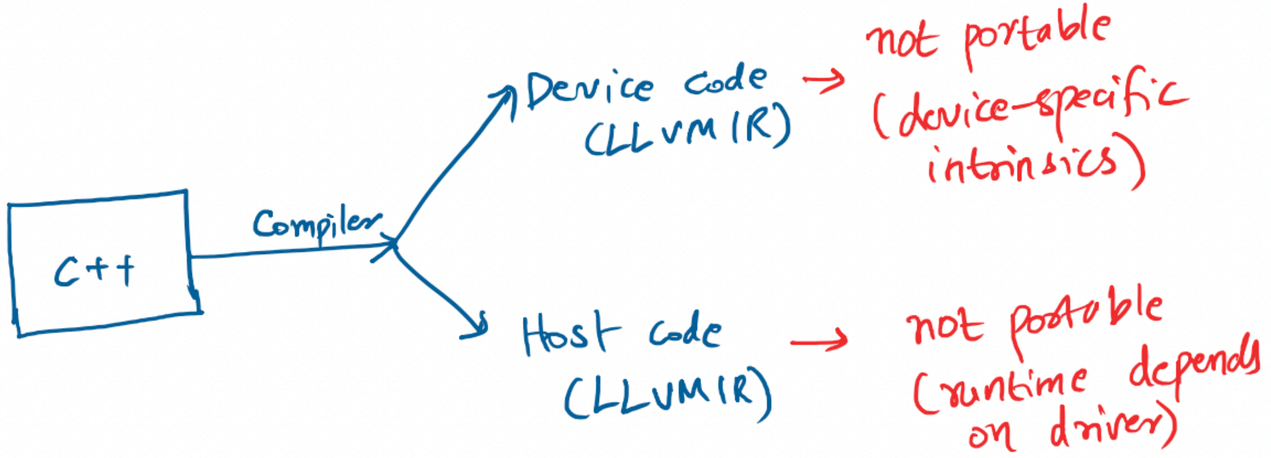
But, We have LLVM?

Unfortunately its design doesn't work for GPUs

Intermediate Representations for GPUs: LLVM Does Not Cut it

Sasank Chilamkurthy | 05 April 2024 | 11 minutes to read.





https://chsasank.com/intermediate-representations-for-gpu.html

Introducing Llama.lisp

My Own Compiler Framework

```
(define ((n-squares void) (n int))
  ; Declare `i` as an integer
  (declare (i int))
  ; Iterate over the first n integers
  (for ((set i 0)
        (lt i n)
        (set i (add i 1)))
       ; `sq` is scoped to the `for` loop body
       (declare (sq int))
       (set sq (mul i i))
       (call print sq))

  (call print sq) ; Error: `sq` is undeclared
  (ret))
```

```
(bril-define ((add5 int) (n int))
    (set (five int) (const 5))
    (set (sum int) (add n five))
    (ret sum))

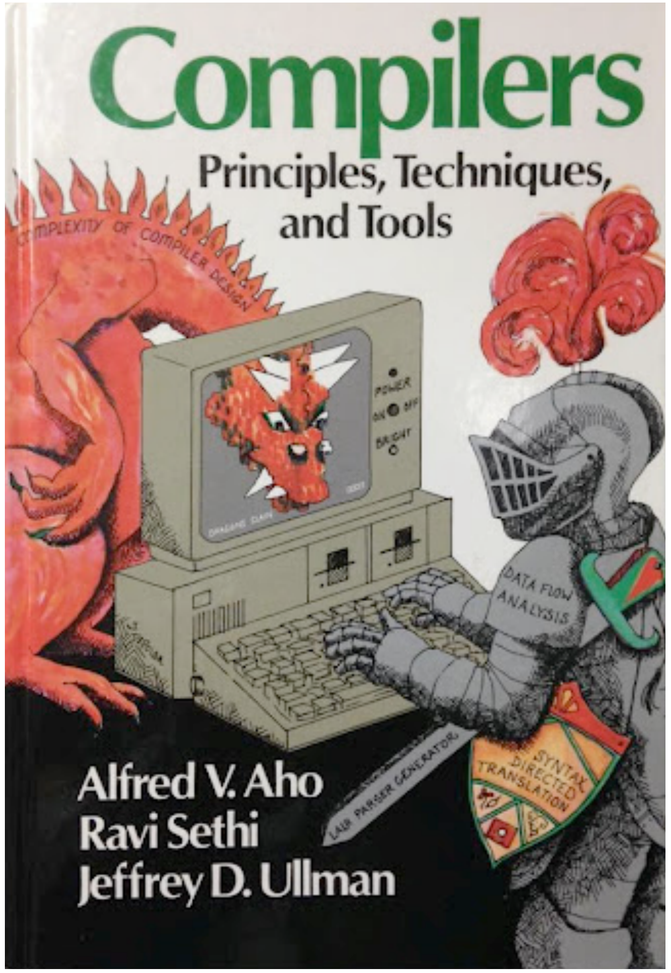
(bril-define ((main int))
    (set (a int) (const 9))
    (set (b int) (call add5 a))
    (ret b))
```

Insides of a Compiler

Compilers are Dragons



Not a pokemon lol

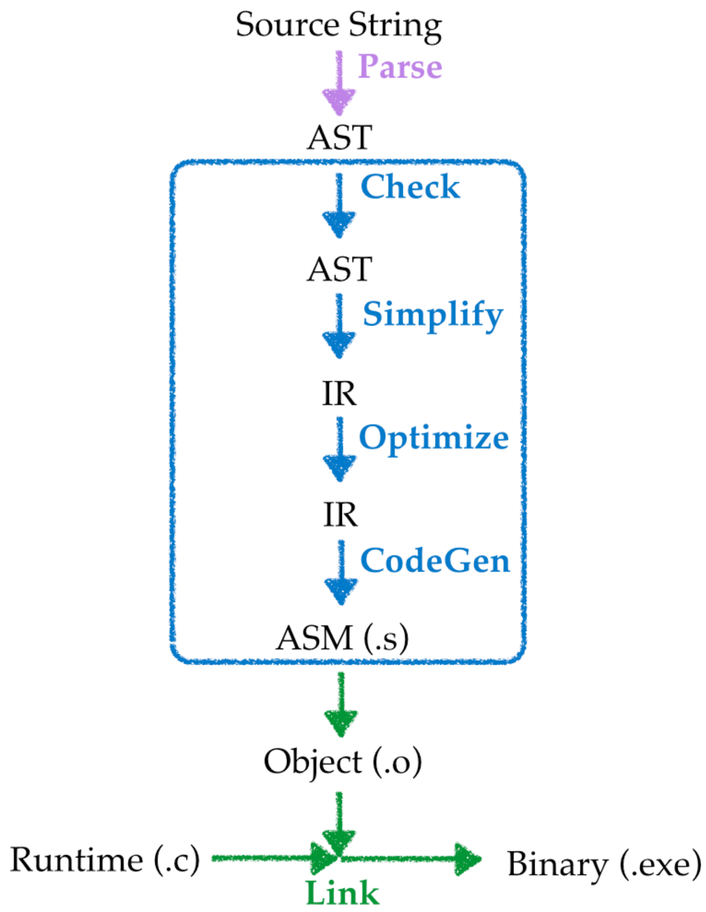




http://xahlee.info/parser/dragon_book_table_of_contents.html

Parts of a Compiler

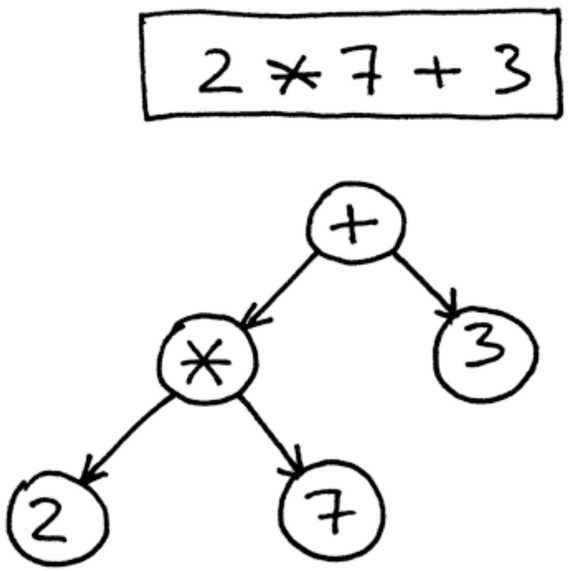
Innards of the dragon

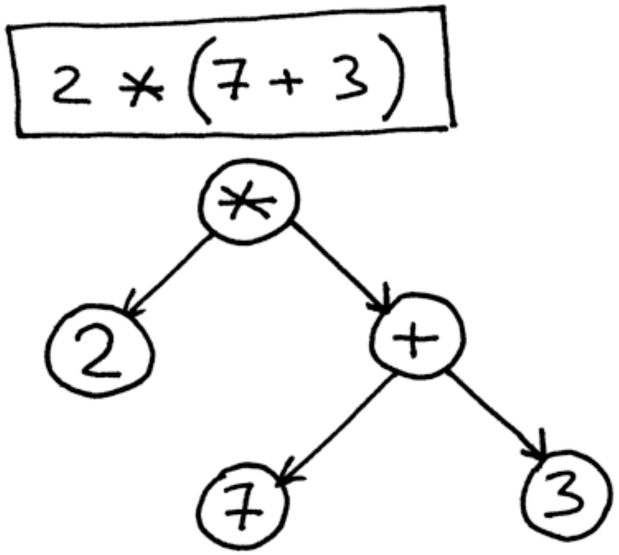


https://www.cs.umd.edu/class/fall2019/cmsc430/Intro.html

Parser

Converts Program to a Abstract Syntax Tree





https://ruslanspivak.com/lsbasi-part7/

Intermediate Representation

Goes closer to assembly

```c
int max(int a, int b) {
  if (a > b) {
    return a;
  } else {
    return b;
  }
}
```

```
define i32 @max(i32 %a, i32 %b) {
entry:
  %retval = alloca i32, align 4
  %0 = icmp sgt i32 %a, %b
  br i1 %0, label %btrue, label %bfalse

btrue:                                            ; preds = %2
  store i32 %a, i32* %retval, align 4
  br label %end

bfalse:                                           ; preds = %2
  store i32 %b, i32* %retval, align 4
  br label %end

end:                                              ; preds = %btrue, %bfalse
  %1 = load i32, i32* %retval, align 4
  ret i32 %1
}
```

Code Generation

IR is transformed to Assembly

```
; ModuleID = ""
target triple = "unknown-unknown-unknown"
target datalayout = ""

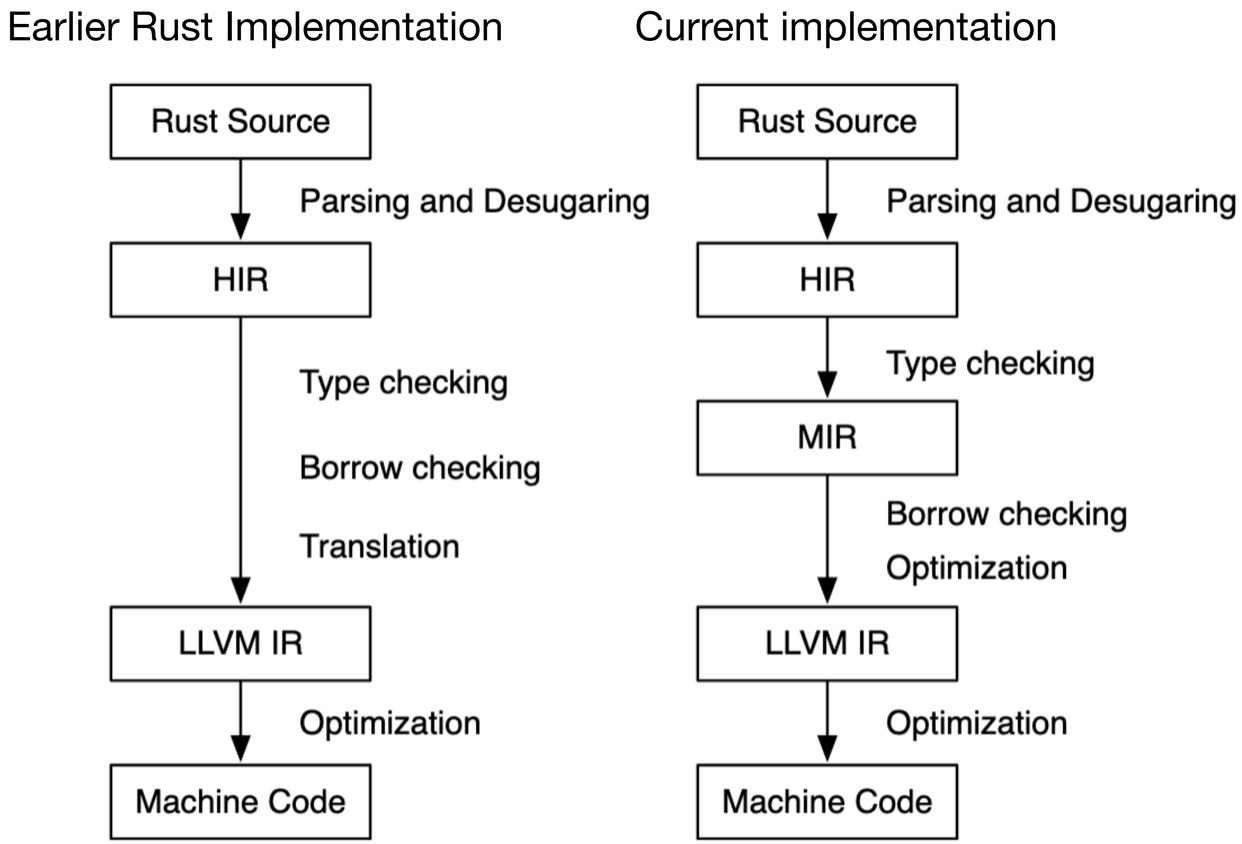
define i32 @"add5"(i32 %"n")
{
entry:
  %"sum" = add i32 %"n", 5
  ret i32 %"sum"
}

define i32 @"main"()
{
entry:
  %"b" = call i32 @"add5"(i32 9)
  ret i32 %"b"
}
```

```
    .text
    .file   "x.ll"
    .globl  add5                            # -- Begin function add5
    .p2align    4, 0x90
    .type   add5,@function
add5:                                   # @add5
    .cfi_startproc
# %bb.0:                                # %entry
    movl    4(%esp), %eax
    addl    $5, %eax
    retl
.Lfunc_end0:
    .size   add5, .Lfunc_end0-add5
    .cfi_endproc
                                        # -- End function
    .globl  main                            # -- Begin function main
    .p2align    4, 0x90
    .type   main,@function
main:                                   # @main
    .cfi_startproc
# %bb.0:                                # %entry
    pushl   $9
    .cfi_adjust_cfa_offset 4
    calll   add5@PLT
    addl    $4, %esp
    .cfi_adjust_cfa_offset -4
    retl
.Lfunc_end1:
    .size   main, .Lfunc_end1-main
    .cfi_endproc
                                        # -- End function
    .section    ".note.GNU-stack","",@progbits
```

Multiple Intermediate Representations

To tame the complexity



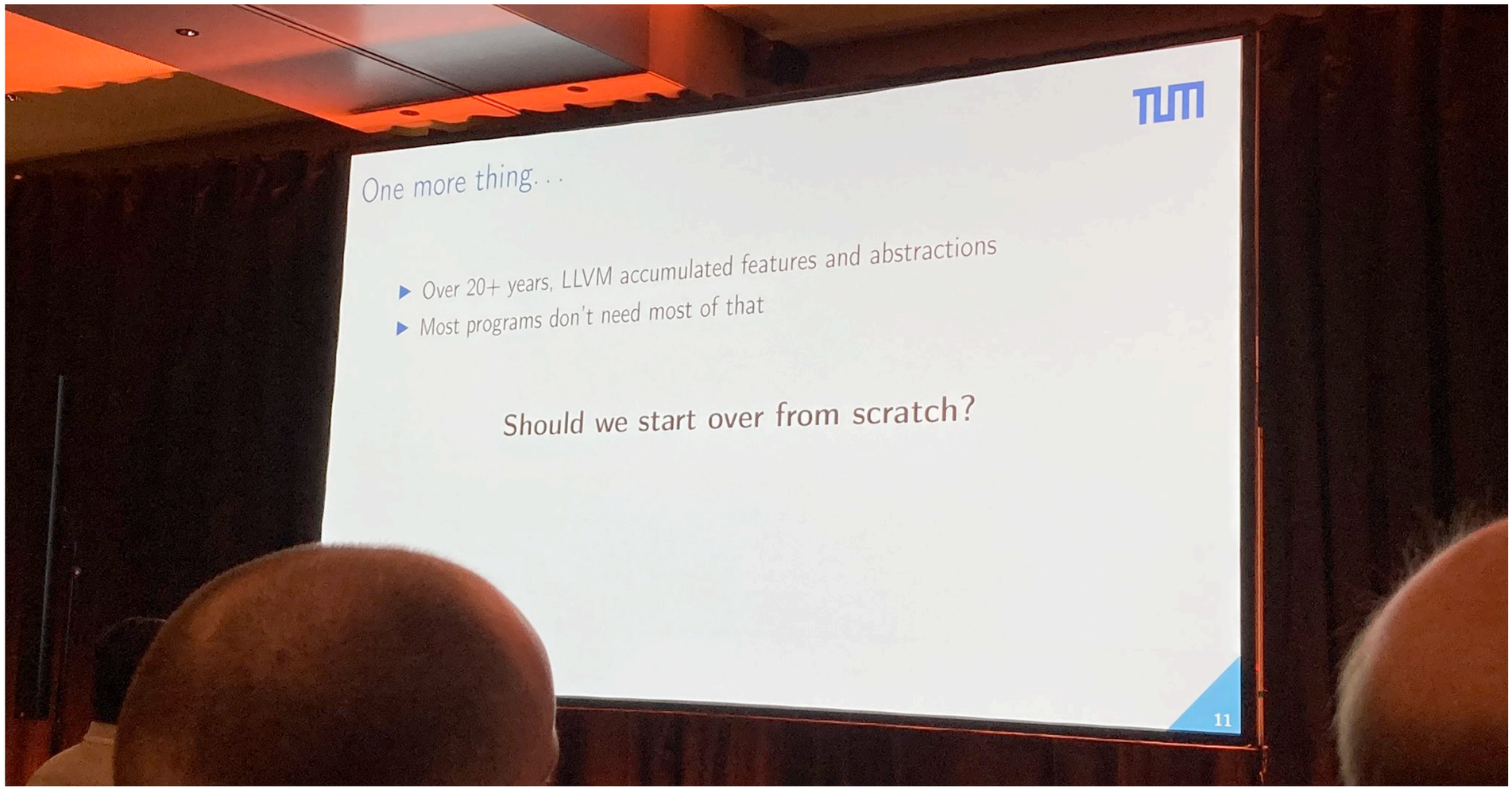
https://blog.rust-lang.org/2016/04/19/MIR.html

Design of llama.lisp

Philosophy

- Bring down the barrier of compiler writing significantly
- Avoid parsing, focus on semantics: JSON as IR
- Use the best tools for a job: implement in any language
- Highly extensible: Layers of languages
- Performance of generated code important

LLVM became too complex



https://x.com/hansw2000/status/1778445240401019178

BRIL: Simplification of LLVM IR



Lessons

Lesson 1: Welcome & Overview
discussion video tasks due **September 9**

Lesson 2: Representing Programs
discussion representing programs getting started with Bril
tasks due **September 14**

Lesson 3: Local Analysis & Optimization
discussion simple dead code elimination local value numbering
tasks due **September 21**

Lesson 4: Data Flow
discussion data flow implementation task
tasks due **September 28**

Lesson 5: Global Analysis & SSA
discussion global analysis & optimization static single assignmer
tasks due **October 7**

Lesson 6: LLVM
discussion introduction to LLVM writing an LLVM pass
tasks due **October 19**

Lesson 7: Loop Optimization
discussion video tasks due **November 2**

Lesson 8: Interprocedural Analysis
discussion video

Lesson 9: Alias Analysis
discussion video

Lesson 10: Memory Management
discussion video tasks due **November 9**

Lesson 11: Dynamic Compilers
discussion Dynamic Compilers Tracing via Speculation
tasks due **December 7**

Lesson 12: Program Synthesis

Bril: A Compiler Intermediate Representation for Learning

- Instruction Oriented
- Minimal and ruthlessly regular
- Language agnostic. Bril programs are just JSON.

https://www.cs.cornell.edu/courses/cs6120/2020fa/

https://capra.cs.cornell.edu/bril/intro.html

BRIL is JSON

So are s-expressions

```
{
  "functions": [
    {
      "name": "add5",
      "args": [{"name": "n", "type": "int"}],
      "type": "int",
      "instrs": [
        { "op": "const", "type": "int", "dest": "five", "value": 5 },
        { "op": "add", "type": "int", "dest": "sum",
          "args": ["n", "five"] },
        { "op": "ret", "args": ["sum"] }
      ]
    },
    {
      "name": "main",
      "args": [],
      "instrs": [
        { "op": "const", "type": "int", "dest": "a", "value": 9 },
        { "op": "call", "type": "int", "dest": "b",
          "funcs": ["add5"], "args": ["a"] },
        { "op": "print", "args": ["b"] }
      ]
    }
  ]
}
```

```
(bril-define ((add5 int) (n int))
    (set (five int) (const 5))
    (set (sum int) (add n five))
    (ret sum))

(bril-define ((main int))
    (set (a int) (const 9))
    (set (b int) (call add5 a))
    (ret b))
```

```
[“bril-define”,[["add5","int"],["n","int"]],
          [“set”,["five","int"],["const",5]],
          [“set”,["sum","int"],["add","n","five"]],
          ["ret","sum"]],
[“bril-define",[["main","int"]],
          ["set",["a","int"], ["const",9]],
          ["set",["b","int"],["call","add5","a"]],
          ["set",["tmp","int"],
          ["call","print","b"]],
          [“ret","b"]]]
```

Pass Around Programs, not Commands

- guile dsl.scm | python llvm.py
- No tight coupling between the different components beyond a JSON schema
- This JSON schema is literally a language
- Every language has got JSON parser, so can use whatever language we want
- Can replace parts easily

Layers of Languages

- **cat ${filename} | guile sexp-json.scm | python c-lisp.py | python brilisp.py | python llvm.py | bash run.sh**
- Simplify the translation by breaking it into layers which interact with each other via messages
- Messages are programs represented in text format

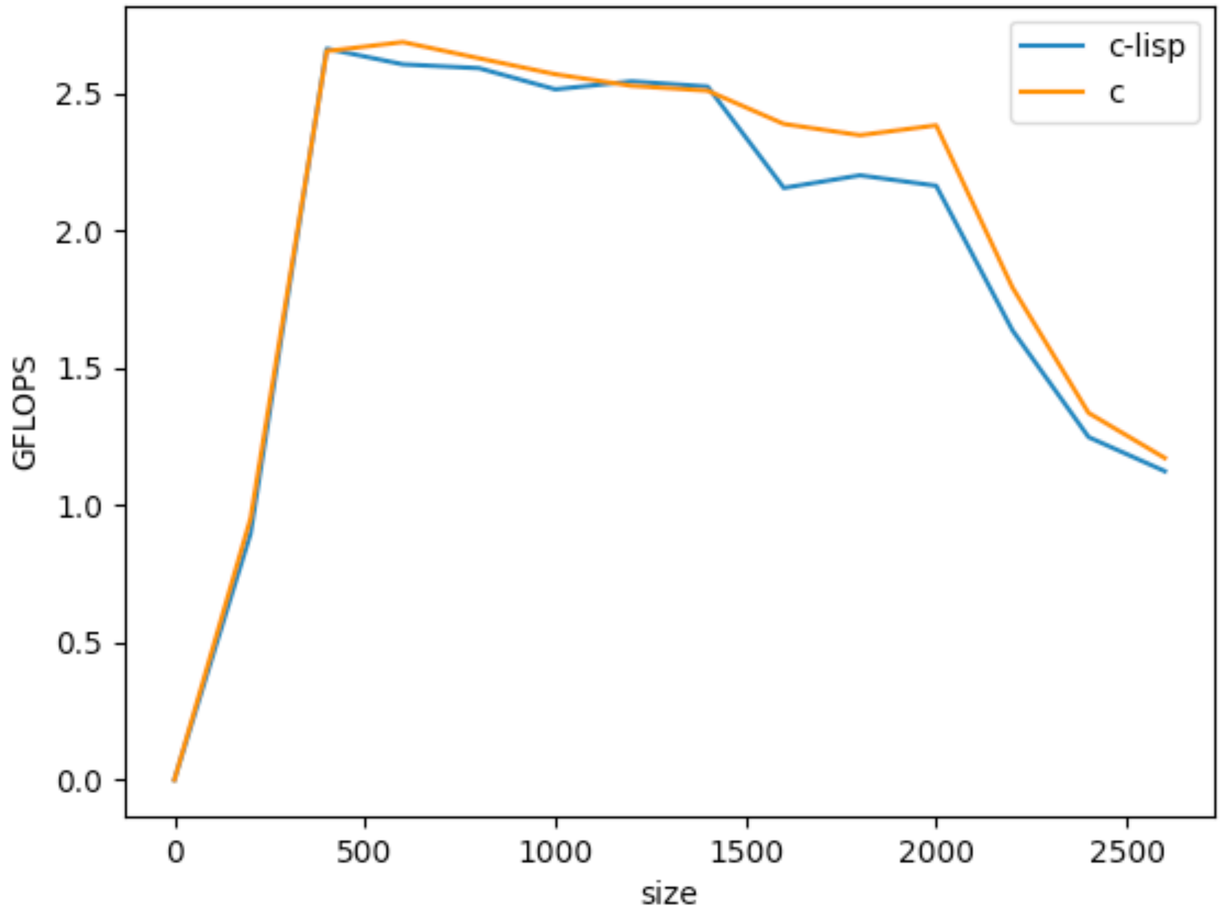
C-Lisp: C Like Frontend

So that we can generate this from other languages!

```
(define ((n-squares void) (n int))
  ; Declare `i` as an integer
  (declare (i int))
  ; Iterate over the first n integers
  (for ((set i 0)
        (lt i n)
        (set i (add i 1)))
       ; `sq` is scoped to the `for` loop body
       (declare (sq int))
       (set sq (mul i i))
       (call print sq))

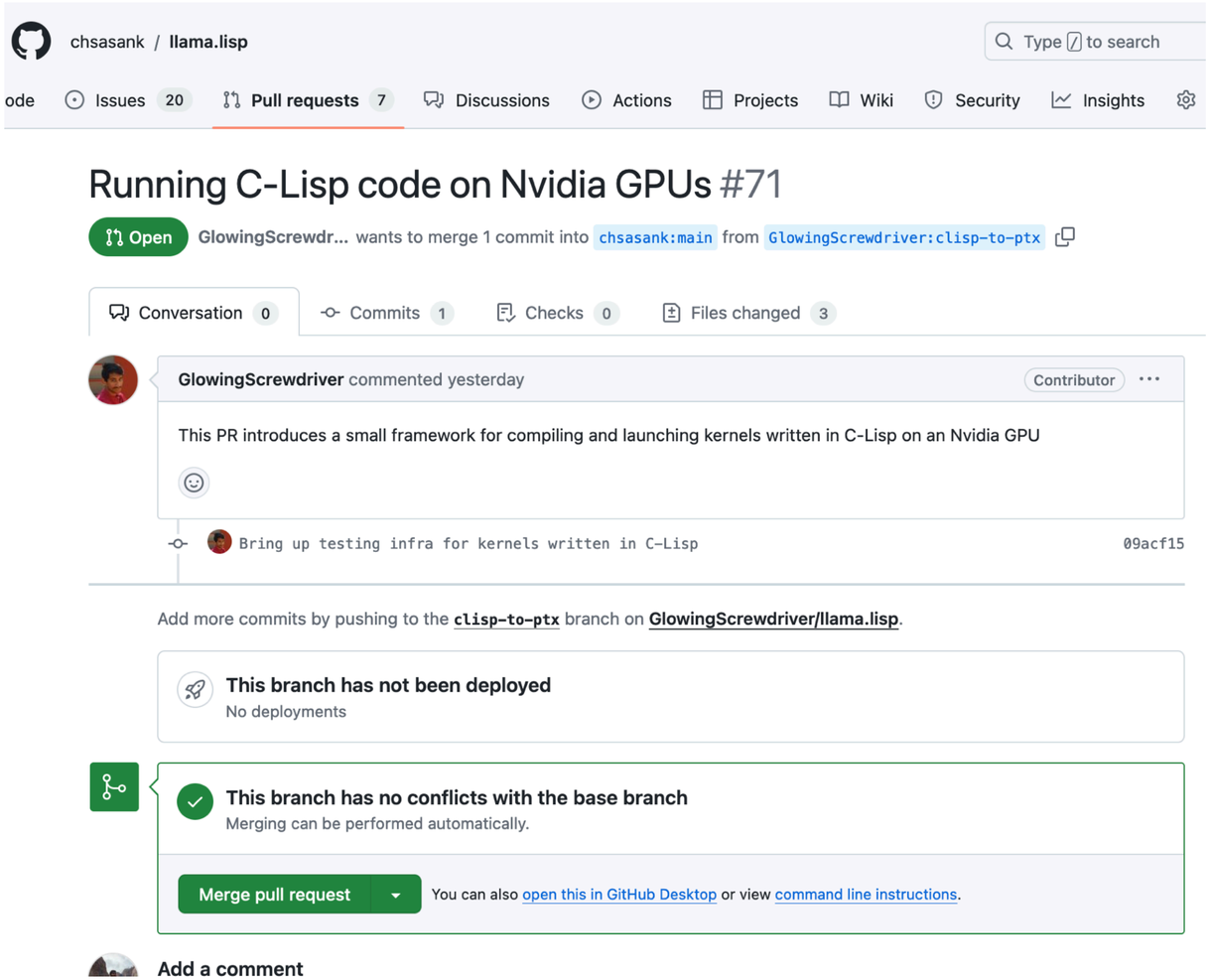
  (call print sq) ; Error: `sq` is undeclared
  (ret))
```

Naive Matrix Multiplication Performance



GPU Support: WIP!

C-Lisp works almost as is!




```
(define ((kernel void) (a (ptr float)) (b (ptr float)) (c (ptr float)) (len int))
        ; Calculate c[i, j] from a[i, *] and b[*, j]
        ; a, b, c have dimensions len x len
        (declare row int)
        (declare col int)
        (set row
             (add
                (call llvm.nvvm.read.ptx.sreg.tid.x)
                (mul (call llvm.nvvm.read.ptx.sreg.ntid.x) (call llvm.nvvm.read.ptx.sreg.ctaid.x))))
        (set col
             (add
                (call llvm.nvvm.read.ptx.sreg.tid.y)
                (mul (call llvm.nvvm.read.ptx.sreg.ntid.y) (call llvm.nvvm.read.ptx.sreg.ctaid.y))))

        (declare a-ptr (ptr float))
        (declare b-ptr (ptr float))
        (declare c-ptr (ptr float))
        (set a-ptr (ptradd a (mul row len)))
        (set b-ptr (ptradd b col))
        (set c-ptr (ptradd c (add col (mul row len))))

        (declare c-val float)
        (set c-val 0.0)

        (declare k int)
        (for ((set k 0)
              (lt k len)
              (set k (add k 1)))
             (set c-val
                  (fadd
                     c-val
                     (fmul (load a-ptr) (load b-ptr))))
             (set a-ptr (ptradd a-ptr 1))
             (set b-ptr (ptradd b-ptr len)))

        (store c-ptr c-val)))
```

Future Work

- Custom GPU Driver
- Kernel generation
- Automatic program generation
- Advanced frontends like FL

Thanks!
Questions?
Please contribute!